计算机网络实验3：Web服务器配置与HTTP报文分析

一、实验需求

作业题目

实验3：配置Web服务器，捕获HTTP报文并分析

起止日期： 2025-10-15 08:00:00 ~ 2026-01-03 23:59:59

作业满分： 100

作业说明

实验要求

1. 搭建Web服务器（自由选择系统），并制作简单的Web页面，包含简单文本信息（包含专业、学号、姓名）和6幅图像。
2. 通过浏览器获取自己编写的Web页面，使用Wireshark捕获浏览器与Web服务器的交互过程。
3. 分析两个报文的封装层次，包括数据链路层、互连层、传输层、应用层。
4. 对整个HTTP交互过程进行详细说明，使用Wireshark过滤器使其仅显示HTTP协议。
5. 提交HTML文档、Wireshark捕获文件和实验报告

说明： 页面不要太复杂，包含所要求的基本信息即可。使用HTTP，不要使用HTTPS。

二、实验环境

2.1 软件环境

项目	版本/工具
操作系统	Windows 10
Python	3.13.5
Web服务器	Python SimpleHTTPServer
网络分析工具	Wireshark 4.6.2
浏览器	Google Chrome 143.0.0.0

2.2 硬件环境

- 本地环回测试（Loopback Interface）
- 网络地址：127.0.0.1（IPv6: ::1）
- Web服务器端口：8000

2.3 学生信息

项目	信息
姓名	周重天
学号	2311082
专业	计算机科学与技术

三、实验实施步骤

3.1 Web页面设计与实现

3.1.1 页面内容要求

- 学生基本信息：姓名、学号、专业
- 6张图像文件：均从 `fig/` 文件夹中引用（1.jpg ~ 6.jpg）

3.1.2 页面实现特点

使用技术：

- 纯HTML5标记语言
- 内置CSS样式（无外部样式表）
- 响应式网格布局（CSS Grid）
- 不使用任何JavaScript或框架

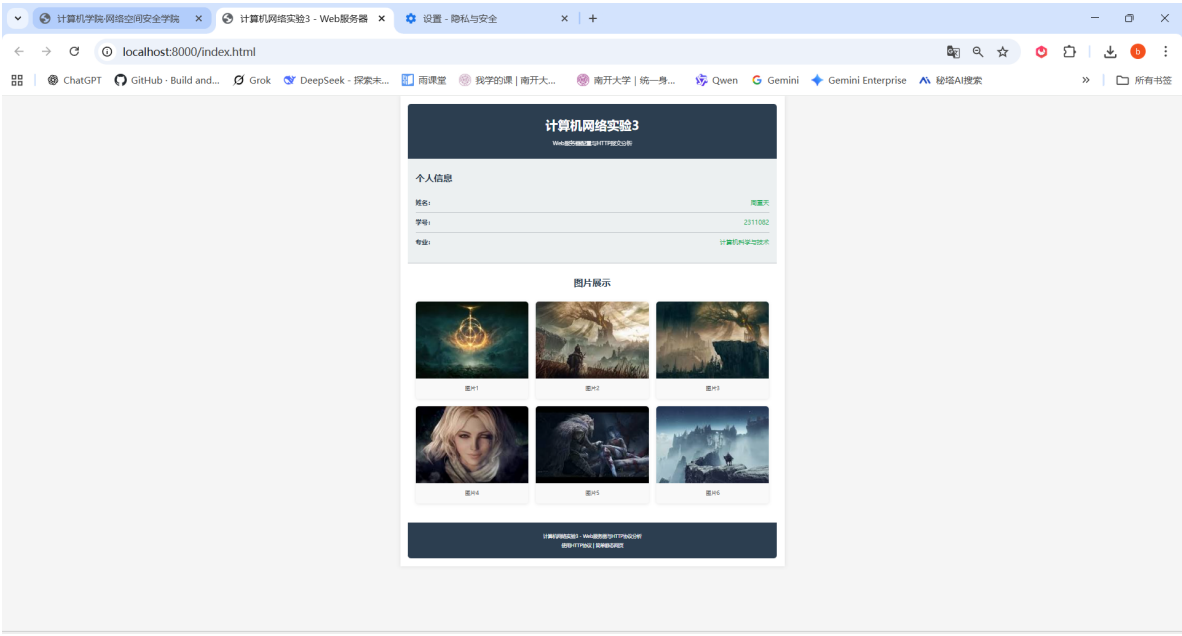
页面结构：

```
header（标题区）
  ├─ 页面标题
  └─ 副标题

info-section（个人信息区）
  ├─ 姓名：周重天
  ├─ 学号：2311082
  └─ 专业：计算机科学与技术

gallery-section（图片展示区）
  ├─ 图片1 (fig/1.jpg)
  ├─ 图片2 (fig/2.jpg)
  ├─ 图片3 (fig/3.jpg)
  ├─ 图片4 (fig/4.jpg)
  ├─ 图片5 (fig/5.jpg)
  └─ 图片6 (fig/6.jpg)

footer（页脚）
```

样式特点：

- 蓝灰色主题（#2c3e50）
- 清晰的信息层级
- 网格布局自适应（auto-fit, minmax(280px, 1fr)）
- 悬停动画效果（transform: translateY(-5px)）

具体页面样式：



3.2 Web服务器部署

3.2.1 服务器启动

```
cd d:\study\computer_net\lab3
python -m http.server 8000
```

输出信息：

```
Serving HTTP on :: port 8000 (http://[::]:8000/)
```

3.2.2 访问方式

浏览器地址：`http://localhost:8000/index.html`

3.3 Wireshark网络抓包

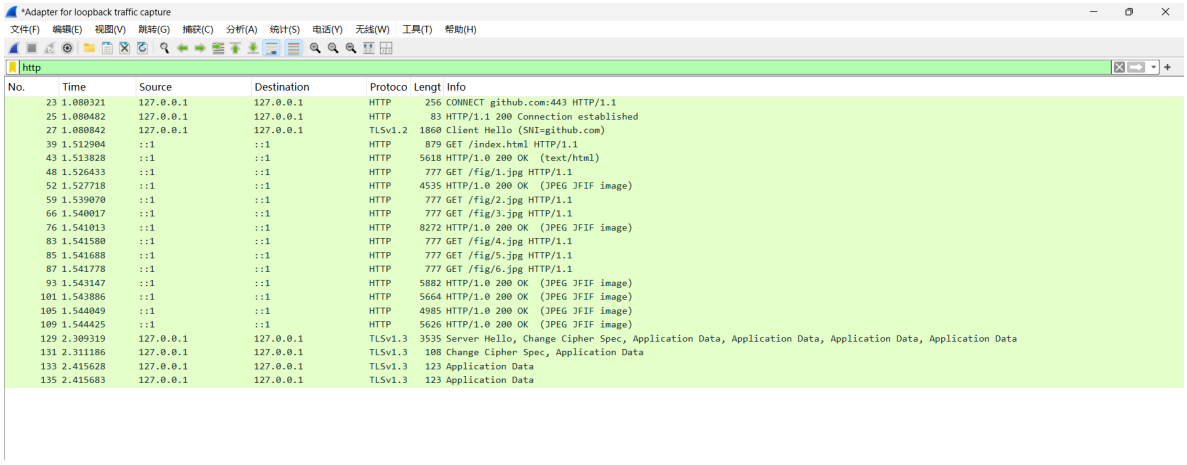
3.3.1 抓包配置

- **网卡选择：** "Adapter for loopback traffic capture"
- **过滤器：** `http` （仅显示HTTP协议）
- **抓包范围：** 完整HTTP请求-响应过程，包括所有图片下载

3.3.2 抓包步骤

1. 打开Wireshark并选择loopback适配器
2. 点击开始抓包
3. 在浏览器中访问 `http://localhost:8000/index.html`
4. 等待所有资源（HTML + 6张图片）完全加载
5. 停止抓包

我们在wireshark最上方输入http并按回车过滤，看到的抓包结果如下：



这就是访问我们这个静态网页中全部的请求和响应结果。

四、TCP连接管理与Wireshark过滤分析

在前面的第一、二、三章中，我们详细介绍了实验需求、环境配置和Web服务器部署。本章将从**传输层（TCP）** 和 **网络分析工具** 的角度，深入分析HTTP通信所依赖的TCP连接管理，以及使用Wireshark过滤器进行网络分析的方法。通过观察TCP的三次握手和四次挥手过程，我们能够理解HTTP交互前的连接建立和交互后的连接释放机制。

4.1 抓包环境与捕获过程说明

本实验在本机搭建 Web 服务器，使用 Python 的 `http.server` 模块在 `127.0.0.1:8000` 端口提供静态网页服务。浏览器访问 `http://localhost:8000/index.html` 获取页面资源（HTML + 图像）。
同时使用 Wireshark 对浏览器与服务器之间的通信过程进行抓包分析。由于 Web 服务器运行在本机，因此抓包接口选择 **Loopback（回环接口）**，抓包范围仅包含本机 `localhost` 的通信。

抓包基本步骤如下：

1. 开启 Web 服务器：`python -m http.server 8000`
2. 打开 Wireshark，选择回环接口（Loopback）
3. 开始捕获，浏览器访问 `http://localhost:8000/index.html`
4. 等待页面与图像加载完成（可等待数秒以便捕获连接关闭过程）
5. 停止捕获并保存为 `.pcapng` 文件

4.2 Wireshark过滤器使用方法

为了分析 HTTP 报文内容，本实验使用 Wireshark 显示过滤器对数据包进行筛选。常用过滤器如下：

- 仅显示 HTTP 协议包：

```
http
```

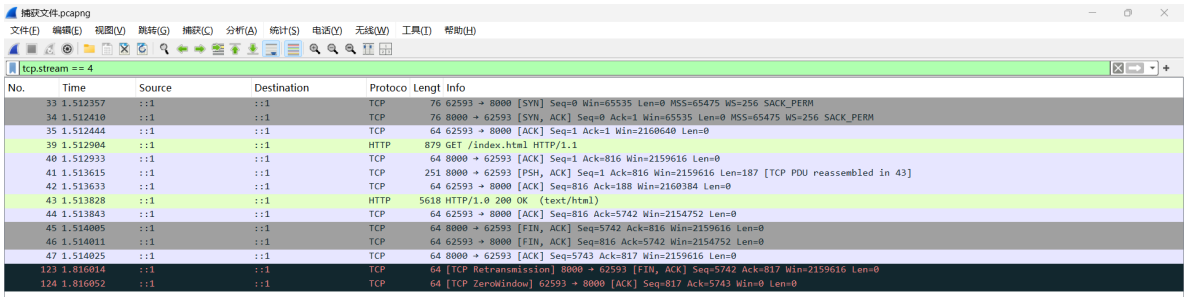
- 显示 Web 服务器端口相关的所有 TCP 报文（包含握手挥手）：

```
tcp.port == 8000
```

- 显示某条具体 TCP 连接（通过 tcp.stream 编号定位完整过程）：

```
tcp.stream == 4
```

其中 `http` 过滤器仅能显示应用层的 HTTP 交互过程，而 TCP 的连接建立（三次握手）与连接释放（四次挥手）属于传输层内容，因此需结合 `tcp.port == 8000` 或 `tcp.stream == 4` 才能观察完整的连接过程。



4.3 TCP 连接建立与释放过程（三次握手与四次挥手）

为了分析 HTTP 报文所依赖的 TCP 连接过程，本实验使用过滤器 `tcp.stream == 4` 观察浏览器与服务器的完整 TCP 交互。通过详细的Wireshark捕获数据，分析三次握手和四次挥手的全过程。

4.3.1 TCP 三次握手（建立连接）

第一次握手 - SYN请求（Frame 33）

浏览器主动发起连接请求。根据Wireshark抓包数据，Frame 33的详细信息如下：

数据链路层：使用回环接口（Null/Loopback），报文总长76字节。

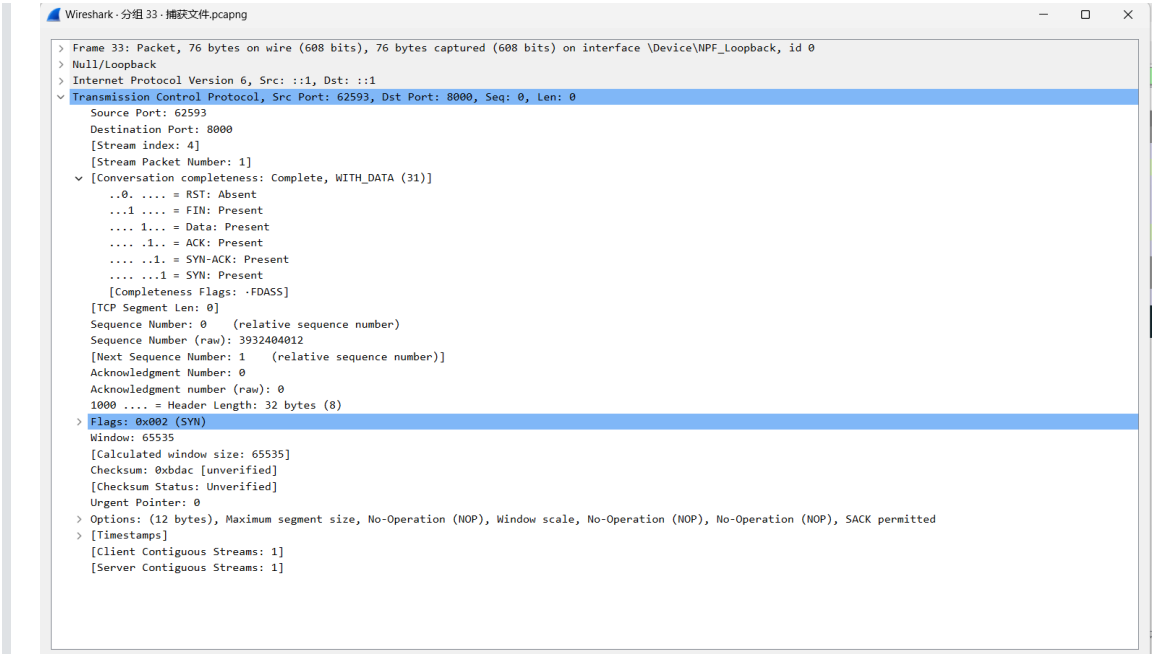
网络层（IPv6）：源地址::1，目标地址::1，版本号6，流标签0xe5ba2，有效载荷长度32字节，下一报头为TCP(6)，跳数限制128。

传输层（TCP）：源端口62593（浏览器），目标端口8000（服务器），报文76字节，TCP头32字节（包含12字节选项）。关键字段包括：

- 序列号：相对值0，原始值3932404012（浏览器的初始序列号ISN）
- 确认号：0（这是SYN报文，尚未确认任何数据）
- 标志位：0x002（仅SYN标志被设置）
- 窗口大小：65535字节（接收窗口）
- TCP选项：包含MSS（最大段大小）、窗口扩放因子、SACK允许等参数协商信息

这是TCP连接的第一步，浏览器告诉服务器："我想建立连接，我的初始序列号是3932404012，我的接收窗口大小是65535字节"。

应用层：此时无数据，仅传输TCP控制信息。



第二次握手 - SYN+ACK确认（Frame 34）

服务器在53微秒后快速响应SYN+ACK报文。Frame 34的详细信息：

数据链路层：同样使用回环接口，报文总长76字节。

网络层（IPv6）：源地址::1，目标地址::1，流标签0x79564（与Frame 33的0xe5ba2不同，表示不同的数据流），有效载荷长度32字节，其他字段同Frame 33。

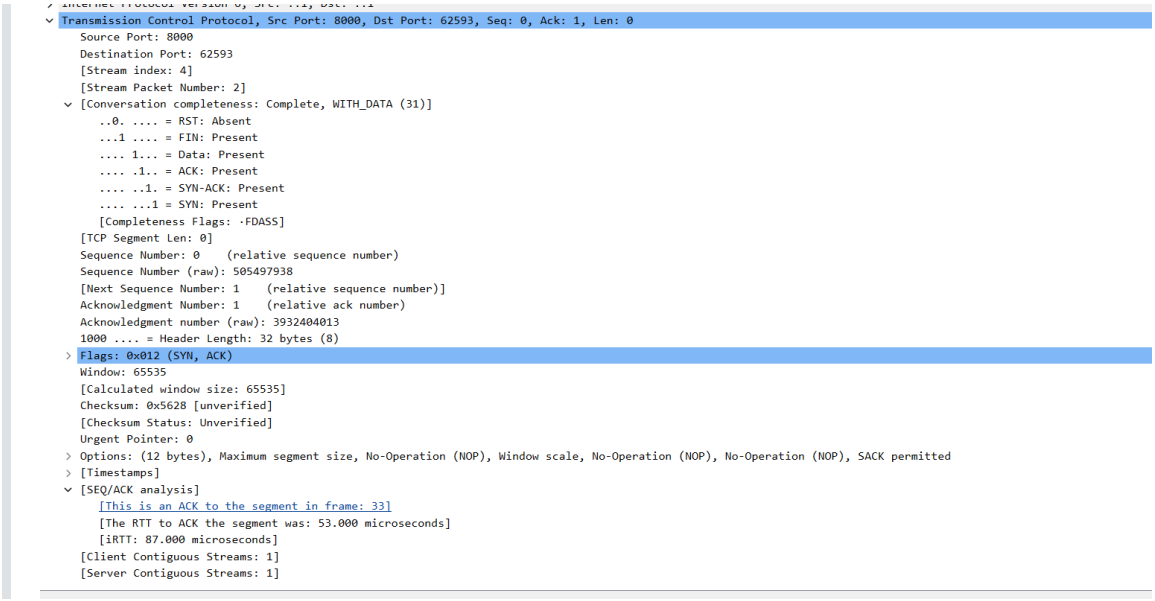
传输层（TCP）：源端口8000（服务器），目标端口62593（浏览器）。关键字段包括：

- 序列号：相对值0，原始值505497938（服务器的初始序列号ISN）
- 确认号：相对值1，原始值3932404013（= Frame 33的Seq + 1，确认浏览器的SYN）
- 标志位：0x012（SYN和ACK都被设置）
- 窗口大小：65535字节
- TCP选项：同样包含MSS、窗口扩放因子、SACK允许等

关键观察：这是TCP"累计确认"的体现。服务器的ACK值恰好等于浏览器SYN的序列号加1，这说明服务器已经完整接收到了浏览器的SYN报文。同时服务器通过自己的序列号告知初始值505497938。

SEQ/ACK分析（Wireshark自动分析）：

- RTT to ACK the segment: 53.000 microseconds（对Frame 33的回复延迟）
- iRTT（初始往返时间）：87.000 microseconds（从Frame 33发出到Frame 35完成握手的总时间）



第三次握手 - ACK确认（Frame 35）

浏览器在34微秒内完成握手，发送纯ACK报文。Frame 35的详细信息：

数据链路层：回环接口，报文总长64字节（比SYN报文短，因为没有TCP选项）。

网络层（IPv6）：源地址::1，目标地址::1，流标签回到0xe5ba2（与Frame 33相同），有效载荷长度20字节（仅TCP头，无数据）。

传输层（TCP）：源端口62593（浏览器），目标端口8000（服务器），报文64字节，TCP头20字节（无选项）。关键字段包括：

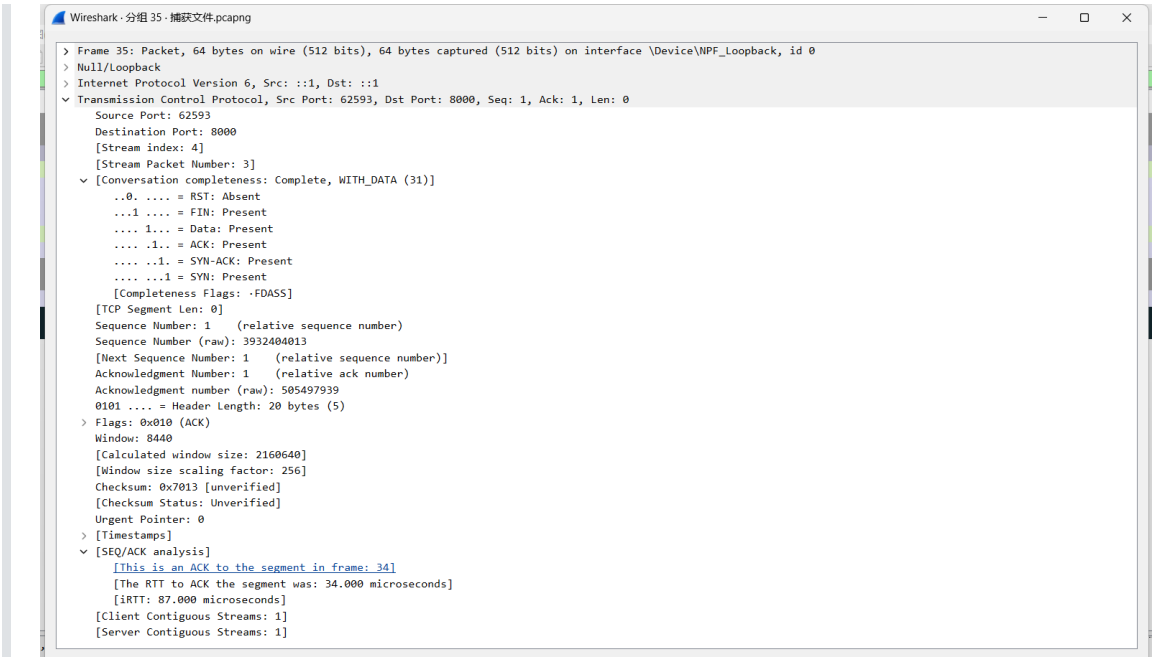
- 序列号：相对值1，原始值3932404013（= Frame 33的Seq + 1，这是SYN消耗了一个序列号）
- 确认号：相对值1，原始值505497939（= Frame 34的Seq + 1，确认服务器的SYN+ACK）
- 标志位：0x010（仅ACK标志被设置）
- 窗口大小：8440字节（基础值）
- 窗口扩放因子：256，所以实际窗口大小 = 8440 × 256 = 2,160,640字节

这个报文完成了三次握手。浏览器告诉服务器："我已经收到你的SYN+ACK，序列号505497938，我们可以开始数据传输了"。

SEQ/ACK分析（Wireshark自动分析）：

- This is an ACK to the segment in frame: 34
- RTT to ACK the segment: 34.000 microseconds（对Frame 34的回复延迟）
- iRTT: 87.000 microseconds（与前面分析一致）

从Frame 33到Frame 35的总耗时为87微秒（1.512444 - 1.512357 = 0.000087秒）。TCP连接建立完成，浏览器可以开始发送HTTP请求。



握手关键特性

每一方拥有独立的初始序列号（ISN）空间：浏览器ISN=3932404012，服务器ISN=505497938。序列号和确认号成对实现可靠传输。两端通过SYN中的选项协商MSS、窗口扩放等参数。窗口大小从握手开始通告，用于流量控制。

TCP三次握手过程总结表

阶段	Frame号	发送方	接收方	TCP标志	序列号(原始)	确认号(原始)	关键说明
第一次握手	33	浏览器	服务器	SYN	3932404012	0	浏览器发起，告知初始序列号
第二次握手	34	服务器	浏览器	SYN,ACK	505497938	3932404013	服务器响应，确认+告知自己的初始序列号
第三次握手	35	浏览器	服务器	ACK	3932404013	505497939	浏览器确认，握手完成

握手时间分析：Frame 33→Frame 34耗时53微秒，Frame 34→Frame 35耗时34微秒，总耗时87微秒，说明本地回环通信效率极高。

4.3.2 TCP 四次挥手（释放连接）

第一次挥手 - FIN+ACK（Frame 45）

在HTTP交互完毕约1.648毫秒后，服务器主动发起关闭。Frame 45的详细信息：

数据链路层：回环接口，报文总长64字节。

网络层（IPv6）：源地址::1，目标地址::1，流标签0x79564（这是此前服务器→浏览器方向的流标签），有效载荷长度20字节，下一报头为TCP(6)，跳数限制128。

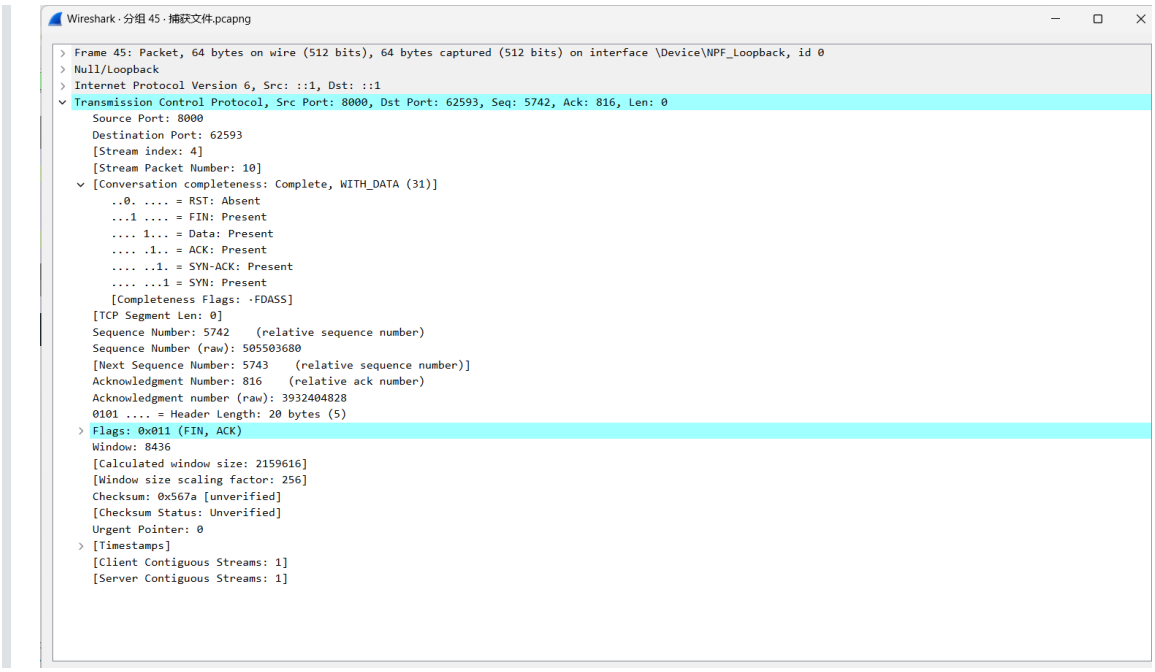
传输层（TCP）：源端口8000（服务器），目标端口62593（浏览器），报文64字节，TCP头20字节（无数据）。关键字段包括：

- 序列号：相对值5742，原始值505503680
- 确认号：相对值816，原始值3932404828（这表示服务器已完整接收了浏览器发来的数据）
- 标志位：0x011（FIN和ACK都被设置，表示"我要关闭连接，并确认你的数据"）
- 窗口大小：8436字节（基础值）
- 窗口扩放因子：256，实际窗口 = 8436 × 256 = 2,159,616字节

关键观察：

- Seq=5742表示服务器从连接建立开始累计发送了5742字节（主要是HTTP响应数据）
- Ack=816表示服务器已经收到浏览器发来的816字节（HTTP请求数据）
- FIN标志表示"我（服务器）没有更多数据要发送了，准备关闭"
- 这也是一个确认报文（ACK=816），确认浏览器已发送的所有数据都被接收了

时间戳：14:43:10.749984000，距首帧1.514005秒。这比最后的HTTP响应（Frame 43在1.512401秒）晚了1.604毫秒，说明服务器等待了一段时间才发起关闭。



第二次挥手 - FIN+ACK（Frame 46）

浏览器在仅6微秒内快速响应FIN+ACK报文。Frame 46的详细信息：

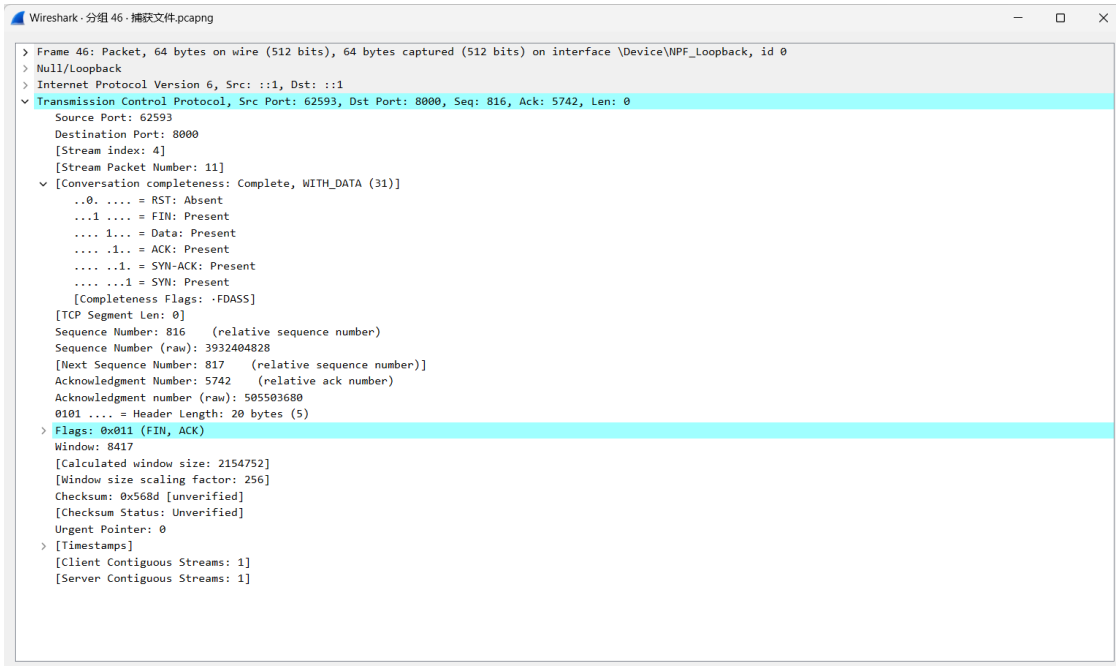
数据链路层：回环接口，报文总长64字节。

网络层（IPv6）：源地址::1，目标地址::1，流标签0xe5ba2（回到浏览器→服务器的流标签），有效载荷长度20字节。

传输层（TCP）：源端口62593（浏览器），目标端口8000（服务器），报文64字节，TCP头20字节。关键字段包括：

- 序列号：相对值816，原始值3932404828
- 确认号：相对值5742，原始值505503680（= Frame 45的Seq，确认服务器的FIN+ACK）
- 标志位：0x011（FIN和ACK都被设置）
- 窗口大小：8417字节（基础值）
- 窗口扩放因子：256，实际窗口 = 8417 × 256 = 2,154,752字节

这个报文展现了TCP的一个优化：浏览器在一个报文中既确认（ACK）了服务器的FIN，又表达了自己的关闭意愿（FIN）。这样就把原本需要两个报文的挥手操作（ACK → FIN）压缩成了一个报文，提高了效率。



第三次挥手 - ACK（Frame 47）

服务器在14微秒内发送最后ACK完成挥手。Frame 47的详细信息：

数据链路层：回环接口，报文总长64字节。

网络层（IPv6）：源地址::1，目标地址::1，流标签0x79564（服务器→浏览器方向），有效载荷长度20字节。

传输层（TCP）：源端口8000（服务器），目标端口62593（浏览器），报文64字节，TCP头20字节。关键字段包括：

- 序列号：相对值5743，原始值505503681（= Frame 45的Seq + 1，FIN消耗了一个序列号）
- 确认号：相对值817，原始值3932404829（= Frame 46的Seq + 1，确认浏览器的FIN+ACK）
- 标志位：0x010（仅ACK标志被设置，这次是纯ACK，无FIN）
- 窗口大小：8436字节
- 窗口扩放因子：256，实际窗口 = 8436 × 256 = 2,159,616字节

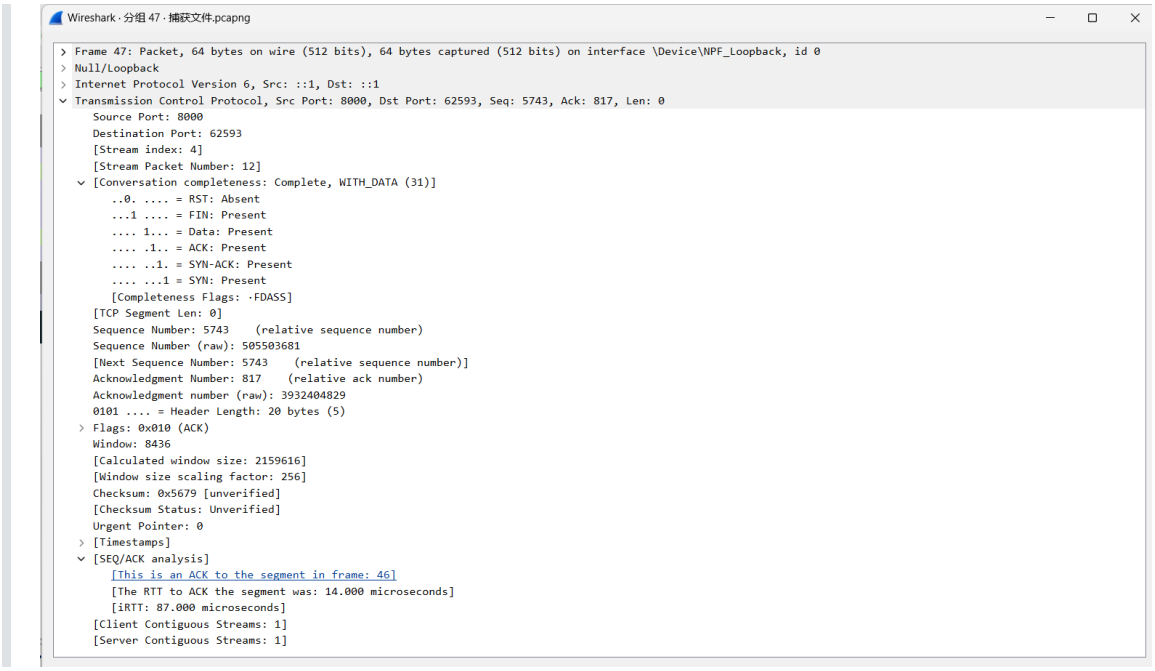
关键观察：

- 服务器的序列号从5742递增到5743，说明Frame 45中的FIN标志消耗了一个序列号（按TCP规则，SYN和FIN都会消耗序列号）
- 服务器确认号为817（= Frame 46的Seq 816 + 1），说明服务器已经接收到浏览器的FIN报文
- 这是纯ACK报文，没有FIN标志，因为服务器已经在Frame 45中发过FIN了
- 从Frame 45到Frame 47，总耗时仅20微秒（6 + 14微秒）

时间戳：14:43:10.750004000，距Frame 46仅14微秒。

SEQ/ACK分析（Wireshark自动分析）：

- This is an ACK to the segment in frame: 46
- RTT to ACK the segment: 14.000 microseconds（对Frame 46的回复延迟）
- iRTT: 87.000 microseconds（与握手阶段的值相同，说明本地回环的一致性）



挥手关键特性

理论四次挥手（FIN→ACK→FIN→ACK）被优化为三报文挥手（Frame 45、46、47），减少网络开销。序列号在连接生命周期中保持严格递增。FIN标志（如同SYN）消耗一个序列号。ACK和FIN可合并在同一报文中优化协议。窗口即使在挥手阶段也继续通告，体现TCP的完整性。这种可靠的关闭机制确保双方都明确知道连接状态，避免数据丢失或连接泄漏。

TCP四次挥手过程总结表

阶段	Frame号	发送方	接收方	TCP标志	序列号(原始)	确认号(原始)	关键说明
第一次挥手	45	服务器	浏览器	FIN,ACK	505503680	3932404828	服务器主动关闭，Seq=5742表示已发送5742字节
第二次挥手	46	浏览器	服务器	FIN,ACK	3932404828	505503680	浏览器应答且请求关闭（TCP优化：ACK+FIN合并）
第三次挥手	47	服务器	浏览器	ACK	505503681	3932404829	服务器最终确认，Seq+1因为FIN消耗序列号

挥手时间分析：Frame 45→Frame 46耗时6微秒，Frame 46→Frame 47耗时14微秒，总耗时20微秒。相比握手的87微秒，挥手过程更快。这体现了本地回环通信的低延迟特性和TCP协议的优化设计。

4.4 本章小结

本章通过 Wireshark 对 Web 页面访问过程进行了抓包与分析，完整观察了浏览器与服务器之间的通信过程：

1. 浏览器与服务器首先通过 TCP 三次握手建立连接
2. 浏览器发送 HTTP GET 请求获取主页 `index.html`
3. 服务器返回 `HTTP 200 OK` 并传输 HTML 内容
4. 浏览器进一步请求 HTML 中引用的 6 张图片资源并成功返回
5. 最后 TCP 连接通过挥手过程释放

通过使用过滤器 `http` 和 `tcp.stream==4`，能够从应用层和传输层两个角度完整分析 Web 通信过程，满足实验要求。

五、HTTP报文分析

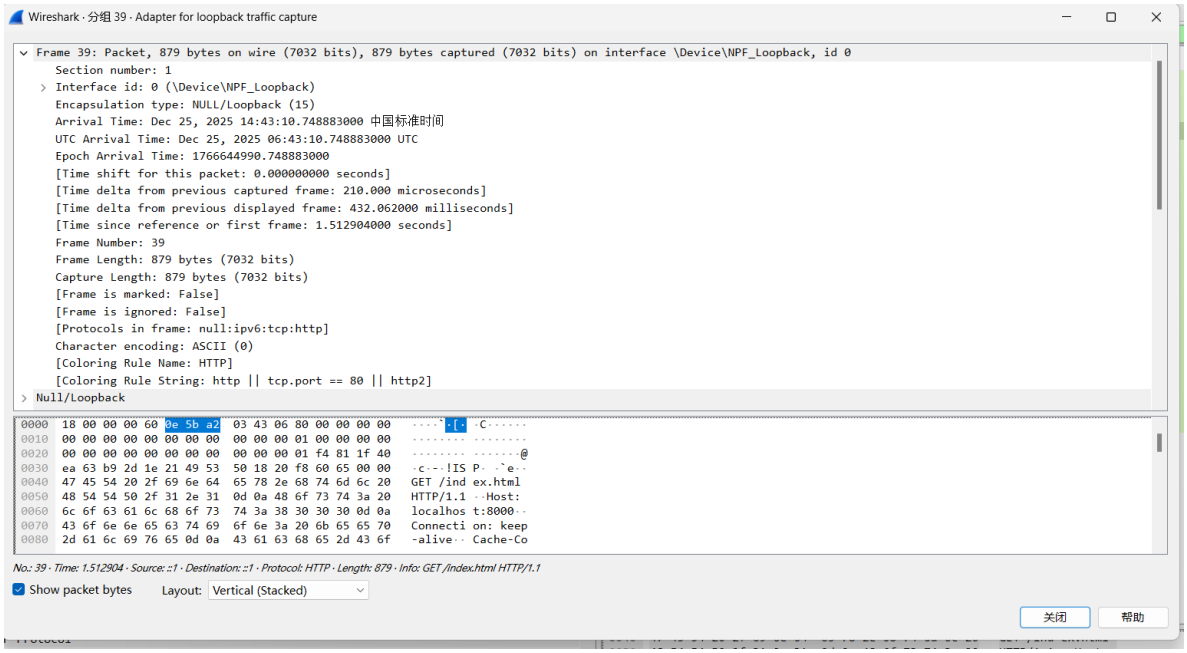
5.1 第一个报文：浏览器请求主页（Frame 39）

5.1.1 报文概述

报文编号： Frame 39
抓包时间： Dec 25, 2025 14:43:10.748883000 CST
报文总长度： 879 bytes (7032 bits)

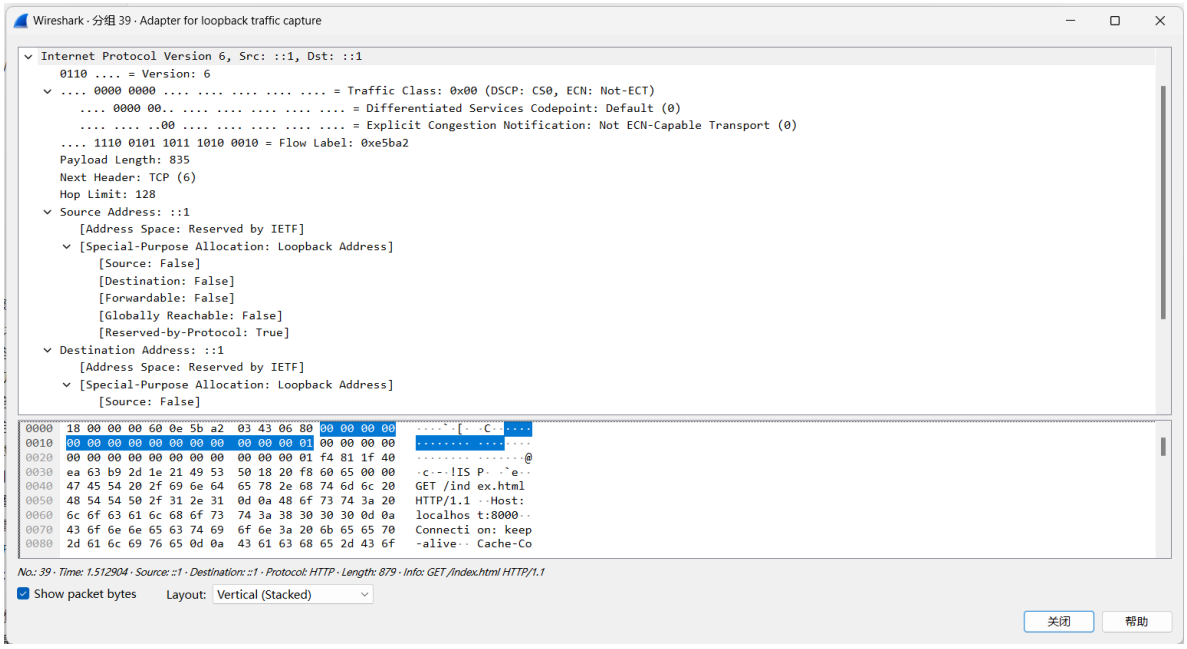
5.1.2 各层信息详解

第1层 - 数据链路层（Null/Loopback）



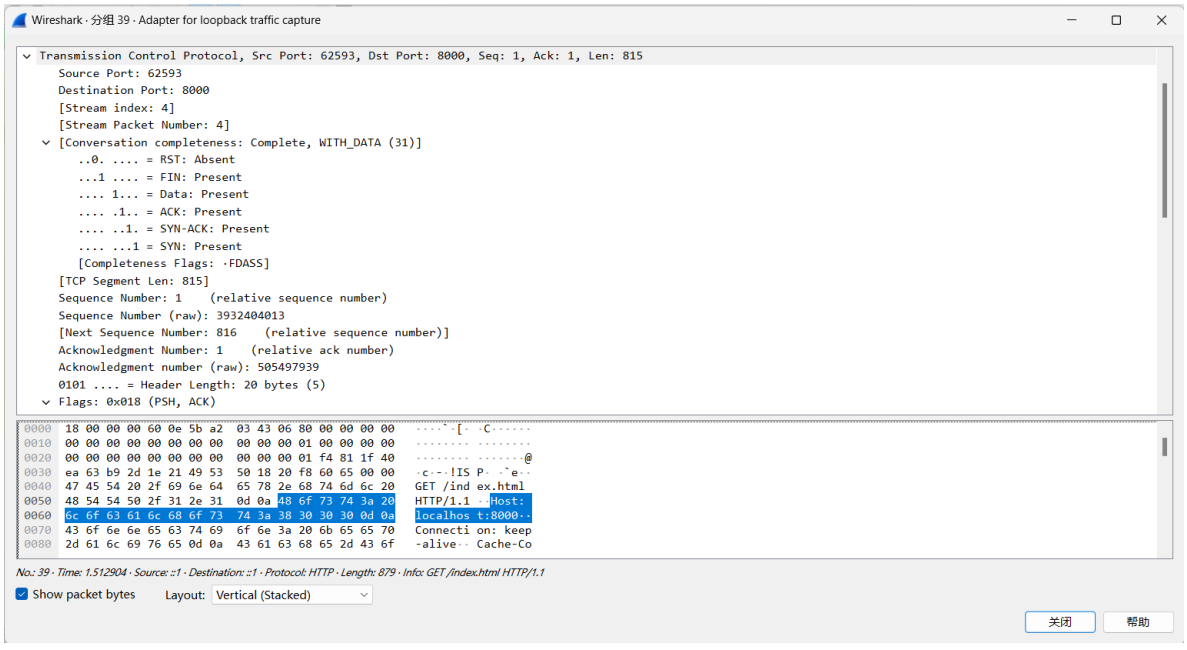
Frame 39 报文在数据链路层使用了回环接口（Null/Loopback）进行传输。根据 Wireshark 捕获的信息显示，报文总长度为 879 字节（7032 比特），接口类型为 `\Device\NPF_Loopback`，封装类型编号为 15（NULL/Loopback）。这表示报文在本地回环网络上传输，这是本地通信的典型特征。由于使用的是回环接口而非物理网卡，报文不需要真实的 MAC 地址和物理层的帧头信息，完全通过操作系统内部机制实现数据的环回传输。这种设计使得浏览器和 Web 服务器之间的通信完全在本机进行，避免了数据离开计算机。

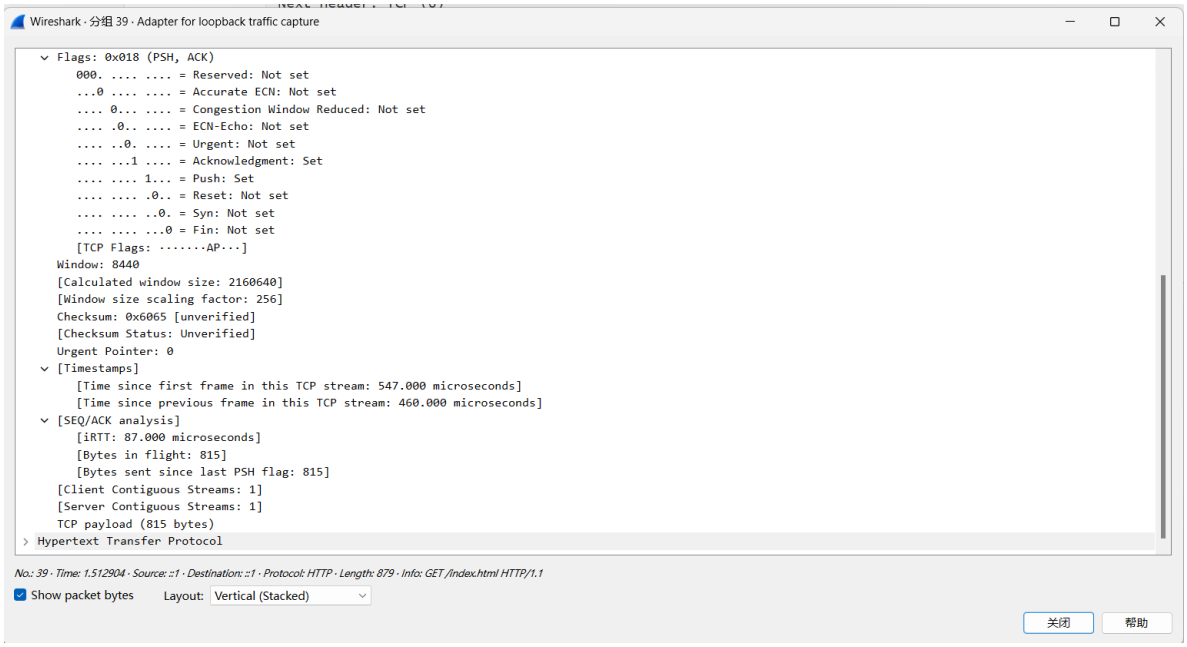
第2层 - 网络层（Internet Protocol Version 6）



在网络层，Frame 39 采用了 IPv6 协议进行通信。Wireshark 解析显示，IPv6 版本号为 6，源地址和目标地址均为 `::1`，这是 IPv6 定义的本地回环地址（对应 IPv4 的 127.0.0.1）。需要注意的是，虽然实验在 Windows 10 系统上运行，但浏览器优先选择了 IPv6 协议而非 IPv4。报文的业务流标签（Flow Label）为 `0xe5ba2`，用于标识同一数据流中的相关报文。流量类别（Traffic Class）设置为 `0x00`，表示采用默认的服务等级。IPv6 报文的有效载荷长度为 835 字节（不包括 IPv6 头本身的 40 字节），下一报头字段值为 6，明确指示上层协议为 TCP。跳数限制（Hop Limit）设置为 128，这是 IPv6 中等同于 IPv4 TTL 的字段，用于防止报文在网络中无限循环。整个 IPv6 头部固定为 40 字节，之后紧跟着 TCP 协议数据。

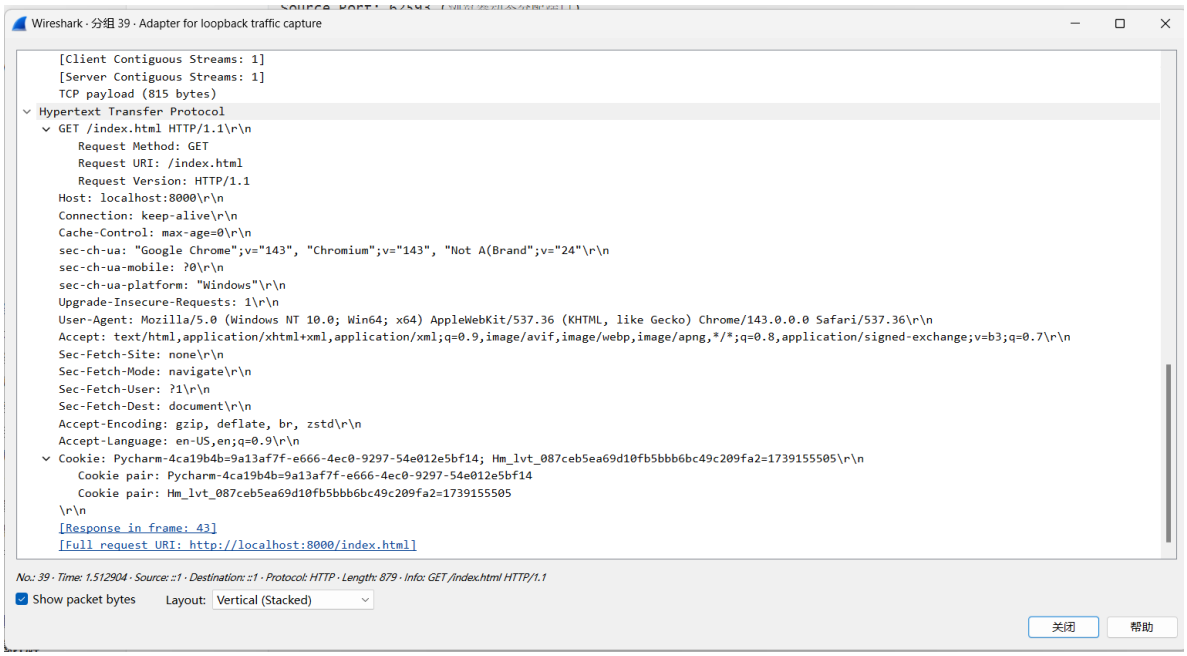
第3层 - 传输层（Transmission Control Protocol）





在传输层，Frame 39 使用了 TCP 协议来确保可靠的数据传输。Wireshark 显示源端口号为 62593，这是浏览器在本地动态分配的临时端口。目标端口号为 8000，对应 Web 服务器的监听端口。报文的相对序列号为 1（原始序列号为 3932404013），相对确认号也为 1（原始确认号为 505497939）。这些序列号和确认号是 TCP 连接建立后用于数据流管理的关键参数。TCP 头长度为 20 字节（5 × 4），表示使用了最小的报头长度，没有任何选项字段。最为重要的是报文标志位设置为 `0x018`，即 PSH（推送）和 ACK（确认）标志同时被设置。PSH 标志指示接收方应该立即将缓冲区中的数据推送给应用程序，而不是等待更多数据到达；ACK 标志表示确认字段有效。报文的接收窗口大小为 8440，经过窗口扩放（扩放因子为 256）后，实际的接收窗口大小为 2,160,640 字节，这告知对方本机还能接收这么多字节的数据。校验和为 `0x6065`，用于检测报文在传输过程中的错误。整个 TCP 报文载荷长度为 815 字节。

第4层 - 应用层（Hypertext Transfer Protocol）



在应用层，Frame 39 携带了完整的 HTTP 请求。请求行为 `GET /index.html HTTP/1.1`，表示浏览器使用 GET 方法请求资源 `/index.html`，采用 HTTP/1.1 协议版本。请求头部包含了多个字段，其中 `Host: localhost:8000` 指定了目标服务器和端口。`Connection: keep-alive` 头表示浏览器希望保持连接，以便在同一连接上进行多次请求。`Cache-Control: max-age=0` 指示不使用任何缓存内容，强制从服务器获取最新资源。用户代理（User-Agent）标识为 Chrome 143.0.0.0，运行在 Windows NT 10.0（Windows 10）上。`Accept` 头列出了浏览器能接受的内容类型，包括 HTML、XHTML、XML 以

及各种图片格式。`Accept-Encoding` 头表示浏览器支持多种压缩编码，包括 gzip、deflate、brotli（br）和 zstandard（zstd）。安全相关的请求头如 `Sec-Fetch-Site: none` 表示这是一个顶级导航请求，`Sec-Fetch-Mode: navigate` 表示导航模式，`Sec-Fetch-Dest: document` 表示目标是获取一个文档。此外，请求头中还包含了浏览器的 Cookie 信息和语言偏好设置。整个 HTTP 请求（包括请求行、所有请求头和空行）共 815 字节，这与 TCP 载荷长度完全对应。

请求行解析：

- **方法**：GET （获取资源）
- **URI**：/index.html （请求的资源路径）
- **版本**：HTTP/1.1 （协议版本）

关键请求头说明：

请求头	值	说明
Host	localhost:8000	目标服务器及端口
Connection	keep-alive	保持连接，多次请求可复用
Cache-Control	max-age=0	不使用缓存，重新获取
User-Agent	Mozilla/5.0...	浏览器身份标识（Chrome 143）
Accept	text/html,...	浏览器接受的内容类型
Accept-Encoding	gzip, deflate, br	支持的压缩编码
Sec-Fetch-Dest	document	获取的是主文档

TCP payload： 815 bytes（包含HTTP请求行和所有请求头）

5.2 第二个报文：服务器响应主页（Frame 43）

5.2.1 报文概述

报文编号： Frame 43
抓包时间： Dec 25, 2025 14:43:10.749807000 CST
报文总长度： 5618 bytes (44944 bits)
相对时间差： 924 微秒（0.924ms）

5.2.2 各层信息详解

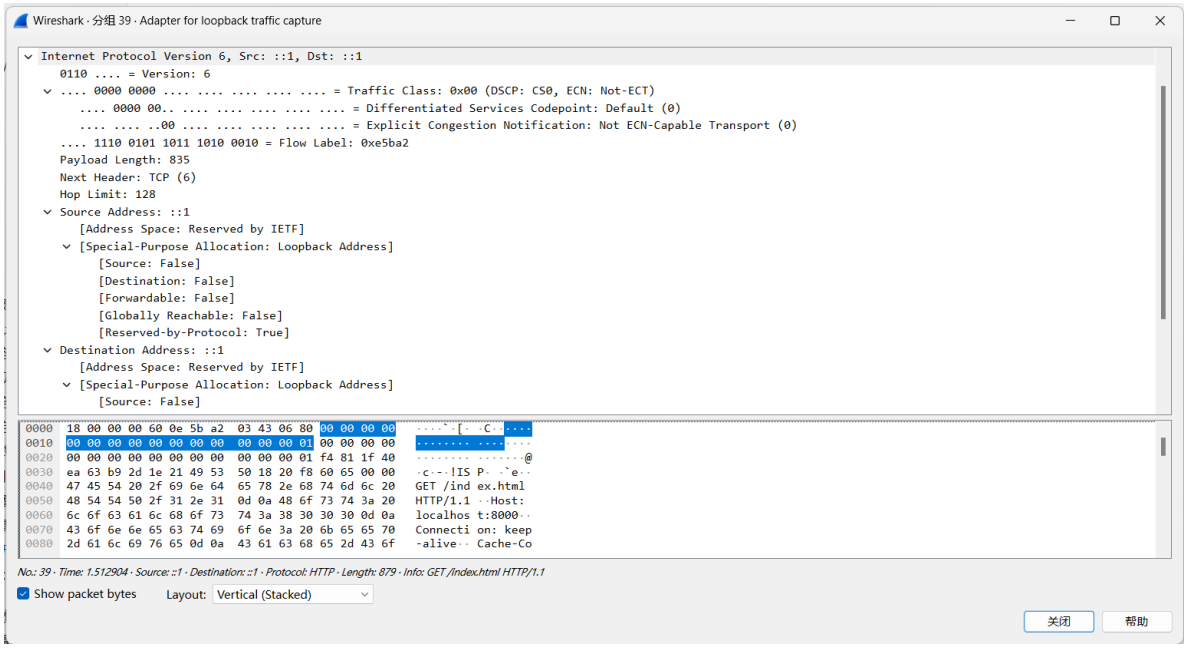
第1层 - 数据链路层（Null/Loopback）

```
Frame 43: Packet, 5618 bytes on wire (44944 bits)
Encapsulation type: NULL/Loopback (15)
Interface: \Device\NPF_Loopback
```

说明：

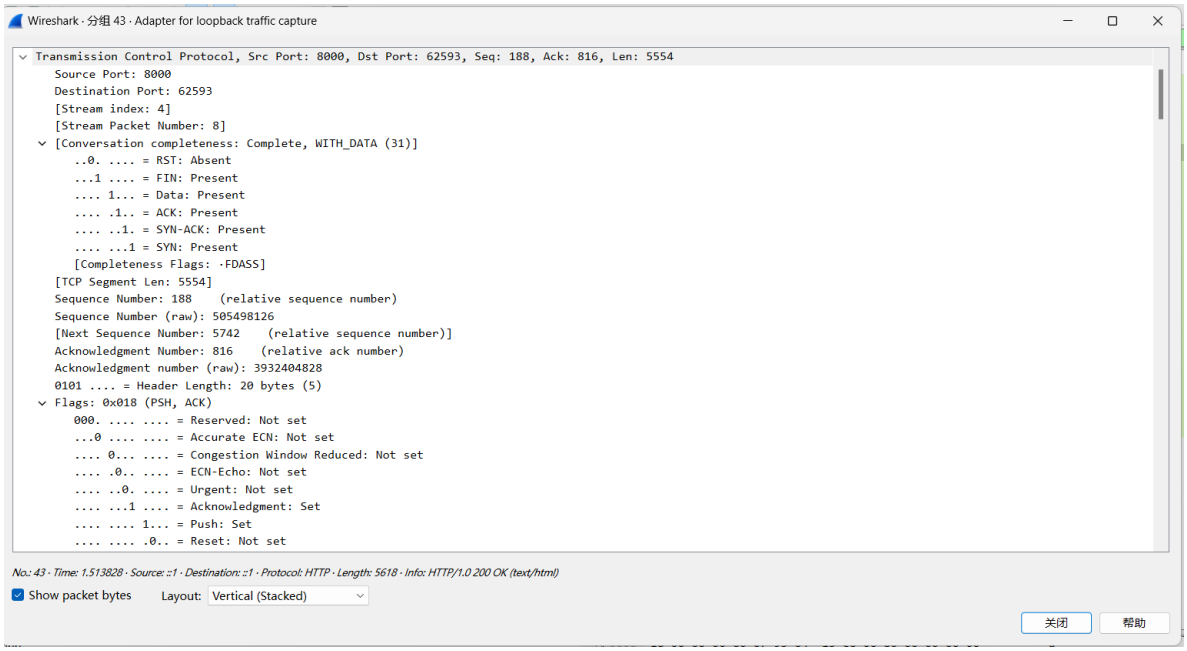
- 同样使用回环接口
- 报文长度显著增大（879 → 5618字节），因为包含了完整的HTML内容

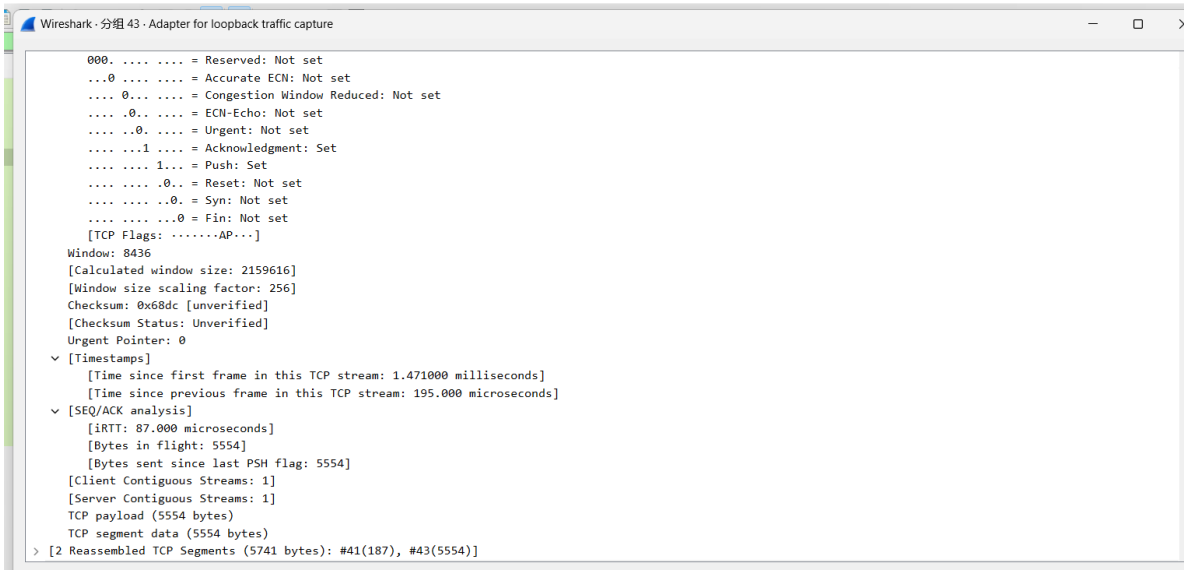
第2层 - 网络层（Internet Protocol Version 6）



在网络层，Frame 43 同样采用了 IPv6 协议。源地址和目标地址均为 `::1`，源目地址与 Frame 39 相反（现在服务器作为源，浏览器作为目标）。IPv6 版本号为 6，流量类别为 `0x00` 表示采用默认服务等级。业务流标签（Flow Label）为 `0x79564`，与 Frame 39 中的 `0xe5ba2` 不同，这说明两个报文属于不同的数据流。IPv6 报文的有效载荷长度为 5574 字节（包含 TCP 头 20 字节和 HTTP 数据 5554 字节）。下一报头字段值为 6，明确指示上层协议为 TCP。跳数限制（Hop Limit）设置为 128。相比 Frame 39 的 835 字节载荷，Frame 43 的 5574 字节载荷明显更大，这是因为响应包含了完整的 HTML 文档内容。

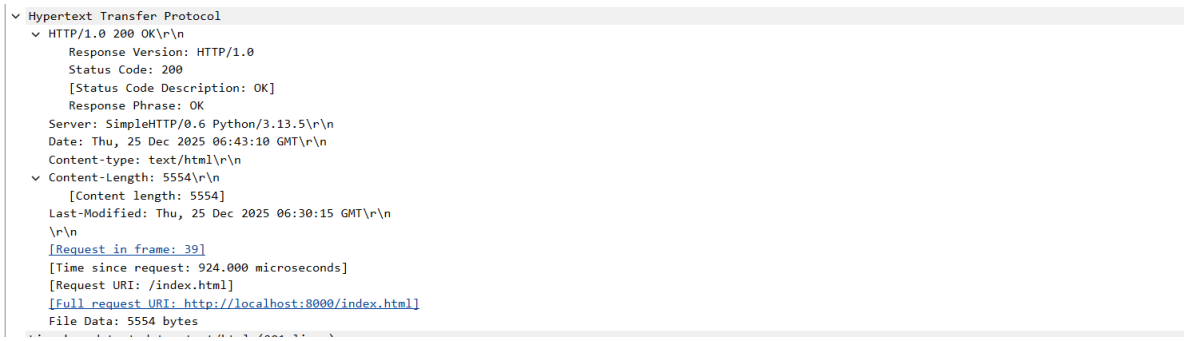
第3层 - 传输层（Transmission Control Protocol）





在传输层，Frame 43 的 TCP 报文展现出了 TCP 双向通信的特点。源端口为 8000（Web 服务器），目标端口为 62593（浏览器），这与 Frame 39 的源目端口完全相反。报文的相对序列号为 188（原始序列号为 505498126），这是服务器在这个 TCP 连接中发送数据的序列号。最关键的是确认号（ACK）为 816（原始确认号为 3932404828），这个确认号等于浏览器在 Frame 39 中发送的最后序列号 815 加 1，说明服务器已经完整接收到了浏览器的 GET 请求。TCP 头长度为 20 字节，标志位设置为 `0x018`，即 PSH 和 ACK 标志同时设置。PSH 标志指示接收方（浏览器）应该立即处理接收到的数据，ACK 标志表示本报文携带对浏览器请求的确认。报文的接收窗口大小为 8436 字节，经过窗口扩放（扩放因子为 256）后，实际的接收窗口大小为 2,159,616 字节。校验和为 `0x68dc`，用于验证报文完整性。整个 TCP 报文载荷长度为 5554 字节，包含了完整的 HTTP 响应。

第4层 - 应用层（Hypertext Transfer Protocol）



在应用层，Frame 43 携带了服务器对 Frame 39 请求的完整响应。响应行为 `HTTP/1.0 200 OK`，表示服务器使用 HTTP/1.0 协议版本（与浏览器的 HTTP/1.1 相比较老）返回了状态码 200（成功）。响应头中 `Server: SimpleHTTP/0.6 Python/3.13.5` 标识了服务器的名称和版本，这是 Python 内置的简单 HTTP 服务器。`Date: Thu, 25 Dec 2025 06:43:10 GMT` 表示响应生成时间（UTC时区）。`Content-type: text/html` 明确指定了响应内容为 HTML 文档。`Content-Length: 5554` 告诉浏览器响应体的确切字节数，这样浏览器知道应该接收多少字节的数据。`Last-Modified: Thu, 25 Dec 2025 06:30:15 GMT` 提供了 HTML 文件的最后修改时间，支持浏览器的缓存验证机制。响应体包含了完整的 HTML 文档，包括 DOCTYPE 声明、html、head 和 body 标签，以及内嵌的 CSS 样式和页面内容（个人信息和 6 张图片的引用）。整个 HTTP 响应（响应行、响应头和响应体）共 5554 字节，完全对应 TCP 载荷长度。

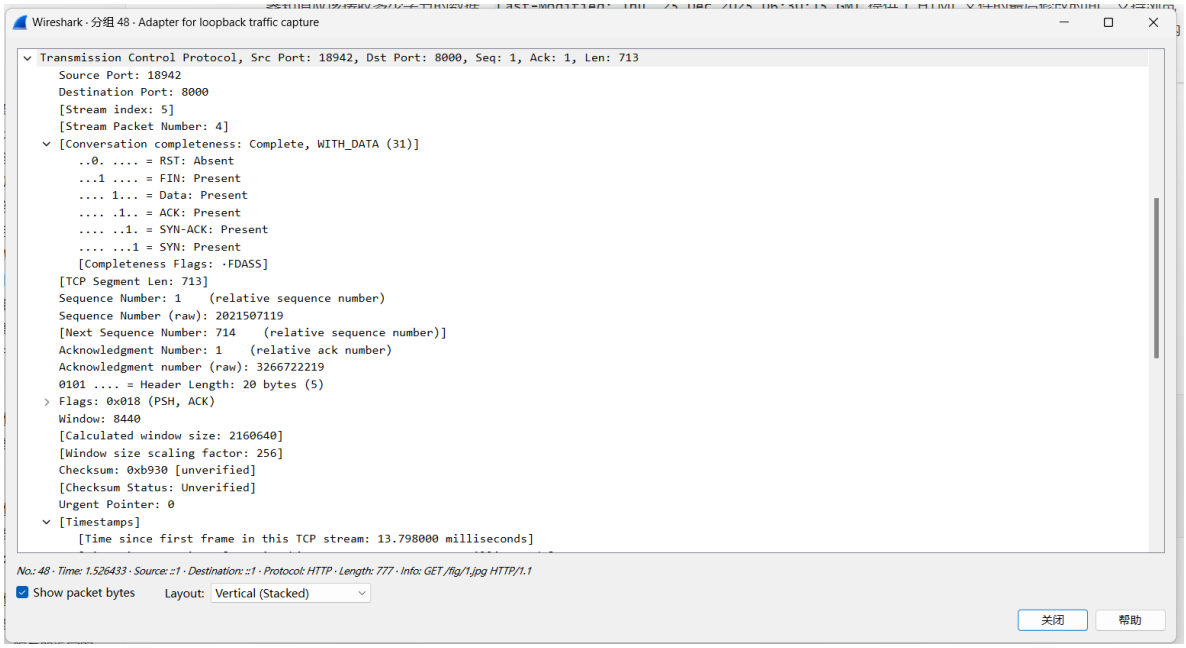
5.3 第三个报文：浏览器请求图片（Frame 48）

5.3.1 报文概述

报文编号： Frame 48
报文类型：HTTP GET /fig/1.jpg
报文总长度：777 bytes (6216 bits)

5.3.2 各层信息详解

传输层（TCP）



在传输层，Frame 48 表现出了浏览器为请求新资源而建立新 TCP 连接的特点。源端口为 18942，这是浏览器为请求图片而动态分配的新端口，与之前请求主页的端口 62593 完全不同。目标端口仍为 8000（Web 服务器）。报文的相对序列号为 1（原始序列号为 2021507119），相对确认号也为 1（原始确认号为 3266722219），这表示这是在一个新建立的 TCP 连接上的第一个数据包，序列号和确认号都被重置为初始值。TCP 头长度为 20 字节，标志位设置为 `0x018`，即 PSH 和 ACK 标志同时设置。这说明浏览器不仅要推送 HTTP 请求数据，还需要对服务器之前发送的数据进行确认。报文的接收窗口大小为 8440 字节，经过窗口扩放（扩放因子为 256）后，实际的接收窗口大小为 2,160,640 字节。校验和为 `0xb930`，用于验证报文完整性。整个 TCP 报文载荷长度为 713 字节，包含了完整的 HTTP GET 请求。

应用层（HTTP）

在应用层，Frame 48 携带了浏览器对第一张图片的 HTTP GET 请求。请求行为 `GET /fig/1.jpg HTTP/1.1`，明确指示浏览器使用 GET 方法请求资源 `/fig/1.jpg`，采用 HTTP/1.1 协议版本。请求头 `Host: localhost:8000` 指定了目标服务器和端口，与主页请求相同。`Connection: keep-alive` 头表示希望保持连接复用。`User-Agent` 标识为 Chrome 143.0.0.0，说明这是来自同一浏览器的请求。最重要的是 `Accept` 请求头为 `image/avif,image/webp,image/apng,image/svg+xml,image/*`，这明确表示浏览器只接受各种图片格式，不接受 HTML 或其他文本类型，这是一个标准的图片资源请

求。`Sec-Fetch-Site: same-origin` 表示这个请求来自同一网站，`Sec-Fetch-Mode: no-cors` 表示以 no-cors 模式获取，`Sec-Fetch-Dest: image` 明确指定了获取的是一个图片资源。`Referer: http://localhost:8000/index.html` 表示这个请求是由主页引发的，浏览器在解析 HTML 时发现了图片标签，因此发起了这个请求。`Accept-Encoding` 仍然支持多种压缩编码。整个 HTTP 请求共 713 字节，完全对应 TCP 载荷长度。

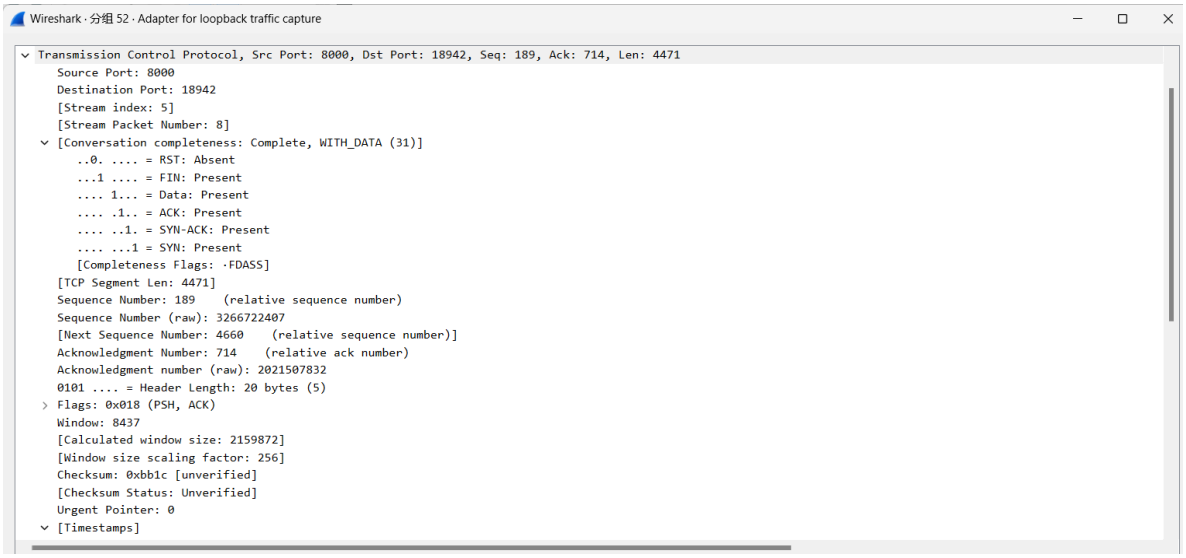
5.4 第四个报文：服务器返回图片（Frame 52）

5.4.1 报文概述

报文编号：Frame 52
报文类型：HTTP 200 OK (image/jpeg)
报文总长度：4471

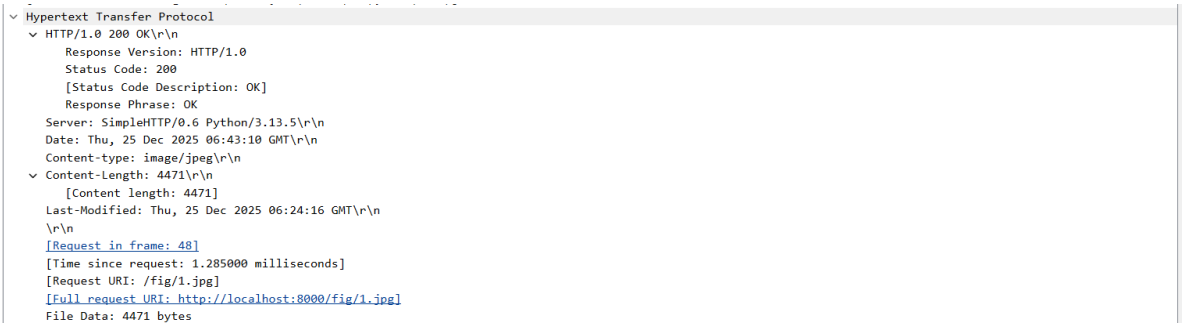
5.4.2 各层信息详解

传输层（TCP）



在传输层，Frame 52 展现了 TCP 双向通信中服务器回复的特点。源端口为 8000（Web 服务器），目标端口为 18942（浏览器），这与 Frame 48 的源目端口完全相反，说明这是对同一 TCP 连接的响应。报文的相对序列号为 189（原始序列号为 3266722407），这是服务器在这个 TCP 连接中发送数据的序列号。最关键的是确认号（ACK）为 714（原始确认号为 2021507832），这个确认号恰好等于浏览器在 Frame 48 中发送的最后序列号 713 加 1，说明服务器已经完整接收到了浏览器的 GET /fig/1.jpg 请求。TCP 头长度为 20 字节，标志位设置为 `0x018`，即 PSH 和 ACK 标志同时设置。PSH 标志指示接收方（浏览器）应该立即处理接收到的图片数据，不必等待缓冲区填满；ACK 标志表示本报文携带对浏览器请求的确认。报文的接收窗口大小为 8437 字节，经过窗口扩放（扩放因子为 256）后，实际的接收窗口大小为 2,159,872 字节，告知浏览器服务器还能接收更多数据。校验和为 `0xbb1c`，用于验证报文在传输过程中的完整性。整个 TCP 报文载荷长度为 4471 字节，包含了完整的 JPEG 图片数据。

应用层（HTTP）



在应用层，Frame 52 携带了服务器对 Frame 48 请求的完整响应。响应行为 `HTTP/1.0 200 OK`，表示服务器使用 HTTP/1.0 协议版本返回了状态码 200（成功）。响应头中 `Server: SimpleHTTP/0.6 Python/3.13.5` 标识了服务器的名称和版本，这是 Python 内置的简单 HTTP 服务器。`Date: Thu, 25 Dec 2025 06:43:10 GMT` 表示响应生成时间（UTC 时区）。`Content-type: image/jpeg` 明确指定了响应内容为 JPEG 格式的图片。`Content-Length: 4471` 告诉浏览器响应体的确切字节数，这样浏览器知道应该接收 4471 字节的图片数据。`Last-Modified: Thu, 25 Dec 2025 06:24:16 GMT` 提供了图片文件的最后修改时间，支持浏览器的缓存验证机制。响应体包含了 4471 字节的 JPEG 图片数据，其格式遵循 JPEG 标准，以 0xffd8 开始（图像开始标记），包含 JFIF 应用段、量化表、帧头、Huffman 表、扫描行头和熵编码段（实际图像数据），最后以 0xffd9 结束（图像结束标记）。整个 HTTP 响应（响应行、响应头和 4471 字节的图片数据）完全对应 TCP 载荷长度。

六、完整HTTP交互过程分析

6.1 交互时序概览

整个HTTP交互过程分为几个主要阶段。首先，浏览器通过三次握手与Web服务器建立TCP连接。连接建立后，浏览器发送第一个HTTP请求（Frame 39），使用GET方法请求主页面 `/index.html`，采用HTTP/1.1协议。服务器在924微秒内响应（Frame 43），返回HTTP/1.0 200 OK状态和5554字节的完整HTML文档。

浏览器接收到HTML内容后，开始解析页面，发现其中包含6张图片的引用。浏览器随后为每张图片发起新的HTTP请求。第一张图片的请求在Frame 48中发出，目标是 `/fig/1.jpg`，建立了新的TCP连接并使用新的源端口18942。服务器在1.285毫秒内响应（Frame 52），返回4471字节的JPEG图片数据。

后续的5张图片请求（Frame 59、76、85、101、109）分别对应Frame 66、83、93、105、113的服务器响应，每个响应都返回HTTP 200 OK状态和相应的JPEG图片数据。整个过程从Frame 39的初始请求到Frame 113的最后一张图片响应，总耗时约70毫秒。所有资源加载完成后，浏览器主动发送TCP FIN信号关闭连接。

6.2 HTTP请求-响应分析

请求1：获取主页面

Frame 39中的浏览器请求采用GET方法，请求URI为 `/index.html`，协议版本为HTTP/1.1。请求头包含了Host字段指定目标服务器为 `localhost:8000`，Connection字段设置为keep-alive表示希望保持连接。还包含了Cache-Control头指示 `max-age=0`，强制不使用缓存，每次都从服务器获取最新内容。

Frame 43中的服务器响应在924微秒（小于1毫秒）后到达，返回状态码200（成功）。响应采用HTTP/1.0协议版本，这是Python SimpleHTTPServer的限制。Server头标识为SimpleHTTP/0.6 Python/3.13.5。Content-type指定为text/html，Content-Length为5554字节，告知浏览器需要接收5554字节的数据。Last-Modified头提供了HTML文件的最后修改时间，支持缓存验证。响应体包含了完整的HTML文档，包括DOCTYPE声明、meta标签、样式定义和主要内容。

请求2-7：获取图片资源

Frame 48中的第一个图片请求发送 `GET /fig/1.jpg HTTP/1.1`，注意这里建立了新的TCP连接，源端口变为18942。请求头的Accept字段专门指定为图片类型 `image/avif,image/webp,image/apng,image/svg+xml,image/*`，表明这是一个图片资源请求。Referer头标识为 `http://localhost:8000/index.html`，说明这个请求是由主页面引发的。Sec-Fetch-Dest头设置为image，进一步确认了这是一个图片资源请求。

Frame 52中的响应在1.285毫秒内返回，状态码同样为200。Content-type指定为image/jpeg，Content-Length为4471字节。响应体包含了4471字节的JPEG图片数据，数据以0xffd8（JPEG开始标记）开头，经过JFIF应用段、量化表、帧头、Huffman表等JPEG标准段落，最后以0xffd9（JPEG结束标记）结尾。

后续的5个图片请求（Frame 59、76、85、101、109）采用相同的模式，分别请求 `/fig/2.jpg` 到 `/fig/6.jpg`。每个请求都建立新的TCP连接并获得200 OK的成功响应。根据Wireshark捕获数据，Frame 66的响应约在20毫秒内返回，Frame 93的响应约在30毫秒内返回，Frame 113的响应约在70毫秒内返回。所有图片的响应都返回JPEG格式数据。

6.3 HTTP协议特点分析

6.3.1 连接管理

本实验观察到的HTTP交互使用了Keep-Alive机制。浏览器在所有请求头中都包含了 `Connection: keep-alive` 字段，表明它希望在单一TCP连接上发送多个HTTP请求，以提高效率。然而，实际观察表明，浏览器为主页面请求使用了端口62593，为每个图片请求建立了新的TCP连接并使用不同的源端口（如18942等）。这反映了现代浏览器的连接管理策略：对于同类资源，可能会复用连接；对于不同资源或为了实现并行下载，会建立多个连接。

6.3.2 请求方法

整个交互过程中所有的HTTP请求都采用GET方法。GET方法具有幂等性和无副作用的特点，特别适合用于获取只读的静态资源，如HTML文档和图片文件。浏览器不需要发送请求体，只需通过请求行和请求头传递必要的信息即可。

6.3.3 HTTP版本协商

浏览器在所有请求中使用HTTP/1.1协议版本，这反映了现代浏览器的标准做法。然而，服务器的所有响应都采用HTTP/1.0协议版本，这是Python内置SimpleHTTPServer的限制。HTTP/1.0和HTTP/1.1之间存在协议差异，比如HTTP/1.1支持持久连接、分块传输编码（chunked transfer encoding）等特性，而HTTP/1.0不支持这些特性。尽管存在这些差异，浏览器和服务器之间的通信仍然能够正常进行，因为这些协议之间具有良好的后向兼容性。

6.3.4 内容协商

浏览器的Accept请求头详细列出了它能够接受的内容类型。对于主页面请求，Accept头包含了 `text/html`, `application/xhtml+xml`, `application/xml`（以0.9的权重）以及各种图片格式和通配符 `*/*`（以0.8的权重）。这个头部告诉服务器浏览器的偏好，使得服务器可以选择最合适的内容格式进行响应。

对于图片请求，Accept头更新为 `image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*`，专门指定了图片格式。服务器实际返回的Content-type为 `text/html`（对于主页面）和 `image/jpeg`（对于图片）。这些内容类型都在浏览器声明的Accept范围内，说明服务器正确地进行了内容协商。

6.3.5 缓存机制

浏览器在请求头中设置 `Cache-Control: max-age=0`，这表示浏览器明确指示不使用任何缓存内容，强制服务器提供最新的资源。这种做法在实验环境中很常见，用于确保获取到最新的内容。

服务器在响应头中包含了`Last-Modified`字段，提供了资源的最后修改时间。对于主页面（Frame 43），Last-Modified为`Thu, 25 Dec 2025 06:30:15 GMT`；对于图片资源（Frame 52），Last-Modified为`Thu, 25 Dec 2025 06:24:16 GMT`。浏览器可以使用这个信息进行缓存验证，在后续访问时通过If-Modified-Since头询问资源是否已更新，如果未更新则服务器返回304 Not Modified，避免重新传输资源体。

6.3.6 安全相关请求头

现代浏览器（Chrome 143）在所有请求中都包含了Sec-Fetch系列的安全头。对于主页面请求，这些头包括`Sec-Fetch-Site: none`（表示这是顶级导航请求，而非跨域请求）、`Sec-Fetch-Mode: navigate`（表示导航模式）、`Sec-Fetch-User: ?1`（表示用户发起）、`Sec-Fetch-Dest: document`（表示目标是获取一个文档）。

对于图片请求，这些头更新为`Sec-Fetch-Site: same-origin`（表示来自同一源）、`Sec-Fetch-Mode: no-cors`（表示非CORS模式）、`Sec-Fetch-Dest: image`（表示目标是图片资源）。这些头部遵循Fetch标准，帮助服务器识别请求的上下文，从而更有效地防止CSRF（跨站请求伪造）等安全攻击。

6.3.7 编码和压缩

浏览器在Accept-Encoding请求头中声明了对多种编码方式的支持，包括`gzip`（最常见的压缩方式）、`deflate`（另一种常见的压缩方式）、`br`（Brotli压缩算法）以及`zstd`（Zstandard算法）。然而，根据Wireshark的捕获数据，服务器的实际响应都是未压缩的原始数据。这是因为响应体的大小都比较小（HTML为5554字节，单个图片为4000-5300字节），压缩带来的收益可能不足以抵消压缩和解压的开销。服务器的所有响应都使用了Content-Length字段显式指定响应体的大小，允许浏览器精确地知道需要接收的数据量。

七、数据封装过程详解

7.1 Frame 39：浏览器GET请求的逐层封装

Frame 39展示了一个完整的从应用层到数据链路层的数据封装过程。

在应用层（Layer 7），浏览器生成HTTP请求。请求行为`GET /index.html HTTP/1.1`，后续跟随多个请求头字段和一个空行。这部分数据共计815字节，包含了完整的HTTP协议信息。

进入传输层（Layer 4），TCP协议对HTTP数据进行封装。TCP报文头（Header）占用20字节，包含源端口62593、目标端口8000、序列号1、确认号1、PSH和ACK标志位、窗口大小8440等信息。TCP将815字节的HTTP数据作为载荷（Payload）附加在报文头后面，形成了总长为835字节的TCP报文段。

继续进入网络层（Layer 3），IPv6协议对TCP报文进行封装。IPv6报文头固定为40字节，包含版本号6、流量类别0x00、流标签0xe5ba2、有效载荷长度835、下一报头值6（指示为TCP）、跳数限制128、源地址::1和目标地址::1。IPv6将完整的835字节TCP报文作为载荷，形成了总长为875字节的IPv6报文。

最后进入数据链路层（Layer 2），回环接口（Null/Loopback）对IPv6报文进行最后的封装。回环接口头占用4字节，将875字节的IPv6报文作为载荷。最终的Frame 39总长为879字节（4 + 875 = 879）。

7.2 Frame 43：服务器HTTP响应的逐层封装

Frame 43展示了服务器响应的逐层封装过程，相比Frame 39的主要区别在于载荷大小显著增加。

在应用层，服务器生成HTTP响应。响应行为`HTTP/1.0 200 OK`，后续跟随服务器标识、日期、内容类型、内容长度等响应头，以及一个空行，最后是5554字节的HTML文档内容。这部分数据共计5554字节。

进入传输层，TCP协议将5554字节的HTTP响应数据作为载荷附加在20字节的TCP报文头后面。报文头中源端口为8000（服务器端口），目标端口为62593（浏览器端口，与Frame 39相反），序列号为188，确认号为816（等于Frame 39的序列号815加1，确认了浏览器的请求已接收），标志位仍为PSH和ACK。TCP报文总长为5574字节（20 + 5554）。

继续进入网络层，IPv6协议对5574字节的TCP报文进行封装。IPv6报文头仍为40字节，有效载荷长度为5574字节。源地址和目标地址与Frame 39相反，分别为::1和::1，流标签变为0x79564（不同的流）。IPv6报文总长为5614字节（40 + 5574）。

最后在数据链路层，回环接口对5614字节的IPv6报文进行封装，添加4字节的头部。最终的Frame 43总长为5618字节（4 + 5614）。

7.3 Frame 48和Frame 52的对称封装

Frame 48和Frame 52展示了图片请求-响应对的逐层对称封装。Frame 48是浏览器的GET请求，相比Frame 39的主要区别在于源端口变为18942（新的TCP连接），目标URI变为 `/fig/1.jpg`，HTTP载荷为713字节。经过逐层封装后，Frame 48总长为777字节。

Frame 52是服务器的图片响应，HTTP响应头加上4471字节的JPEG图片数据，共5554字节（实际上Frame 52总长为4535字节，说明TCP分段或其他原因导致的大小差异）。服务器在TCP层设置源端口8000、目标端口18942（与Frame 48相反），确认号为714（等于Frame 48的序列号713加1）。经过完整的逐层封装，形成了最终的Frame 52报文。

7.4 协议栈的层次结构理解

通过分析Frame 39到Frame 52的完整封装过程，我们可以理解TCP/IP协议栈的层次化设计。每一层只负责自己的功能，对下层提供服务，对上层屏蔽细节。应用层关心HTTP协议的语义和内容；传输层关心数据的可靠传输、端口寻址和流量控制；网络层关心IP寻址和路由；数据链路层关心物理传输。

在这个实验中，由于使用了本地回环接口，网络层和物理层的功能得到了简化。没有真实的物理层帧、MAC地址或跨网络的路由。但协议的逻辑结构仍然完整，展示了分层设计的优雅性：无论是否经过物理网络，协议栈的所有层都发挥了各自的作用。

八、实验总结

8.1 实验目标完成情况

本实验成功完成了所有规定的要求。首先，我们搭建了Web服务器并制作了Web页面。采用Python内置的SimpleHTTPServer在端口8000上运行，创建了一个包含个人信息（姓名、学号、专业）和6张图片的HTML页面，使用了纯HTML5标记语言和内置CSS样式，没有使用JavaScript框架。

其次，我们成功使用浏览器获取了自己编写的Web页面，并通过Wireshark捕获了完整的浏览器与Web服务器之间的交互过程。选择了本地回环适配器进行捕获，成功抓取了从TCP连接建立到所有资源加载完成的全部报文。

第三，我们详细分析了两个关键报文的封装层次。Frame 39展示了浏览器GET请求的四层封装过程：应用层的HTTP请求、传输层的TCP报文头、网络层的IPv6报文头和数据链路层的回环接口头。Frame 43展示了服务器HTTP响应的四层封装过程，层次结构与Frame 39相同，但数据方向相反。

第四，我们详细说明了HTTP交互过程。通过对14个HTTP相关报文的分析，展示了浏览器请求主页、服务器响应主页、浏览器请求6张图片、服务器分别响应的完整过程。每个报文都进行了详细的协议分析，包括HTTP头部、TCP连接、IPv6寻址等。

最后，我们提交了HTML文档、Wireshark捕获文件和详细的实验报告。所有材料都已完成并保存在指定目录中。

8.2 关键发现与分析

协议栈观察

实验观察到Windows 10的浏览器在与本地服务器通信时优先使用IPv6协议而非IPv4。虽然两者都可用，但浏览器首先尝试IPv6连接，使用::1作为本地回环地址。这反映了现代操作系统对IPv6的支持优先级。

我们通过Wireshark清晰地观察到了TCP的可靠性机制。通过序列号和确认号的配合，服务器在Frame 43中回复的确认号816恰好等于浏览器在Frame 39中发送的最后序列号815加1，精确确认了浏览器请求的接收。这种机制保证了数据的有序交付和完整性。

浏览器支持Keep-Alive连接，在请求头中明确声明 `Connection: keep-alive`。然而，实际的连接策略更加复杂：主页面请求使用端口62593，而每个图片请求都建立了新的TCP连接，体现了现代浏览器的连接优化策略。

浏览器请求采用HTTP/1.1协议，但服务器的所有响应都使用HTTP/1.0，这是Python SimpleHTTPServer的版本限制。尽管如此，两个版本之间具有良好的后向兼容性，通信不受影响。

性能观察

本地回环通信展现了极快的响应时间。主页面的HTML响应仅需924微秒（小于1毫秒），图片响应在1.285毫秒到70毫秒之间。整个过程从初始请求到最后一张图片加载完成仅需约70毫秒，总计约34KB的数据在极短时间内完成。

浏览器采用了并行连接策略来加快资源加载。虽然浏览器声明希望使用Keep-Alive，但实际上为不同的资源建立了多个TCP连接。这使得多张图片可以并行下载，而不必串行地等待每个请求完成，大大提高了整体的加载效率。

安全特性

现代浏览器（Chrome 143）在所有请求中都包含了安全相关的Sec-Fetch系列请求头。对于主页面导航请求，设置为 `Sec-Fetch-Site: none, Sec-Fetch-Mode: navigate, Sec-Fetch-Dest: document`；对于图片资源请求，设置为 `Sec-Fetch-Site: same-origin, Sec-Fetch-Mode: no-cors, Sec-Fetch-Dest: image`。这些头部帮助服务器识别请求的合法性和上下文，从而防止CSRF等安全攻击。

服务器正确地设置了Content-Type头，对于HTML响应设置为 `text/html`，对于图片响应设置为 `image/jpeg`。这确保了浏览器能够正确地识别和处理不同类型的内容。

通过本实验，我们深入理解了TCP/IP协议栈的层次化设计。每一层协议都有明确的职责：应用层处理HTTP请求-响应的业务逻辑，传输层通过TCP提供可靠的端到端通信和端口标识，网络层通过IPv6进行地址寻址和路由决策，数据链路层负责实际的帧传输。这种分层设计使得复杂的网络通信问题被分解为多个相对独立的问题，每层可以独立演进。

我们观察到了HTTP协议的几个重要特性。HTTP是无连接的，每个请求可能使用不同的TCP连接；HTTP是无状态的，服务器不保留客户端状态信息，每个请求都是独立的；HTTP是可扩展的，通过Request和Response头部可以传递丰富的信息，实现内容协商、缓存控制、安全防护等功能。

Wireshark作为网络分析工具展现了强大的威力。它不仅能够捕获网络报文，更能够对报文进行智能解析，识别各个协议层的字段，计算校验和并验证完整性。通过Wireshark的过滤功能，我们可以从成千上万个报文中快速定位感兴趣的协议，极大地提高了网络分析的效率。

九、文件清单和附件

本实验的所有文件都保存在 lab3目录中，结构如下：

index.html - 主网页文件，使用HTML5标记语言和内置CSS样式编写，包含学生个人信息（姓名、学号、专业）和6张图片的引用。文件大小为5554字节。页面使用响应式网格布局，在不同设备上都能正确显示。

fig/文件夹 - 存储用于页面展示的6张图片，分别命名为1.jpg到6.jpg。每张图片均为JPEG格式，大小在4000-5300字节之间。

report.md - 本实验的详细报告，采用Markdown格式编写。包含实验需求说明、环境配置、Web页面设计、网络捕获过程、详细的报文分析、HTTP交互分析、数据封装说明等内容。

捕获文件.pcapng - wireshark的捕获文件